Ant Colony Optimization and the Traveling Salesman Problem

Ian Barczewski 15th December 2010

OVERVIEW

- What is ACO?
- Problem Description
- Terminology
- The Algorithms
- ACO in motion
- Analyzing The Algorithm
- Results
- Summary

WHAT IS ANT COLONY OPTIMIZATION?

- Used to find optimal paths inside of a graph and give approximate solutions to optimization problems
- routing problems, assignment problems, scheduling problems, subset problems, network learning, etc.
- Based on ants method of finding food

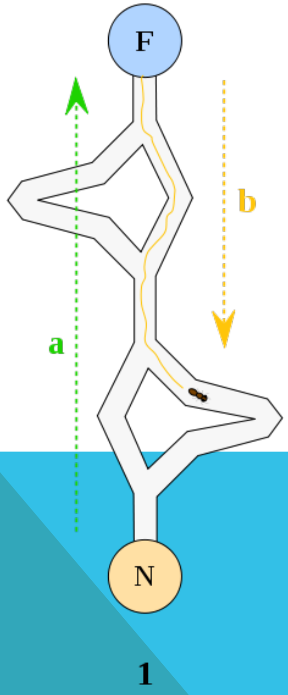


Image Source: Wikipedia

PROBLEM DESCRIPTION

- Ant Colony Optimization uses many variables
  - Ants
  - Cities
- How sensitive is the algorithm to changing these values?
  - Computation Time
  - Cycles
- Relevant when large problems come into play
- Traveling Salesman Problem
  - First problem tested by ACO (Dorigo, 35)
  - Relevant and used in real life applications

Image Source: Wikipedia

TERMINOLOGY

- Pheromone
- Tabu list
- Pheromone evaporation
- Visiblility

THE ALGORITHMS – CHOOSING A CITY

- Each ant has a tabu list
- Next city decided by probability (going from city i to city j)
- J(i, k) are the cities the ant still has to travel to from city i
- n = 1/d(i, j) which is the visibility between the cities i and j
- T(i, j) (t) is the amount of pheromone between cities I and j at time t

$$p_{i,j}^{k}(t) = \frac{\left[\tau_{i,j}(t)\right]^{\alpha} \cdot \left[\eta_{i,j}\right]^{\beta}}{\sum_{l \in J_i^k} \left[\tau_{i,l}(t)\right]^{\alpha} \cdot \left[\eta_{i,j}\right]^{\beta}}$$

THE ALGORITHMS – DEPOSITING PHEROMONE

- Represents each edge (i, j) that the ant visited in iteration t
- Otherwise, it is zero.
- Q is a constant, and L is the cost of the ant’s tour, usually the length, with t representing iteration and k representing the ant

$$\Delta\tau_{i,j}^{k}(t) = \begin{cases} Q / L^{k}(t) \\ 0 \end{cases}$$

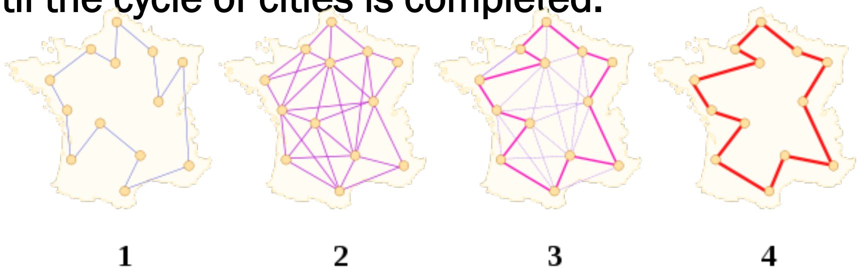
THE ALGORITHMS – PHEROMONE DECAY

- Each edge will have a coefficient p applied to it to represent decay
- M represents the amount of ants in the system

$$\tau_{i,j}(t+1) = (1-\rho)\cdot\tau_{i,j}(t) + \sum_{k=1}^{m}\left[\Delta\tau_{i,j}^{k}(t)\right]$$

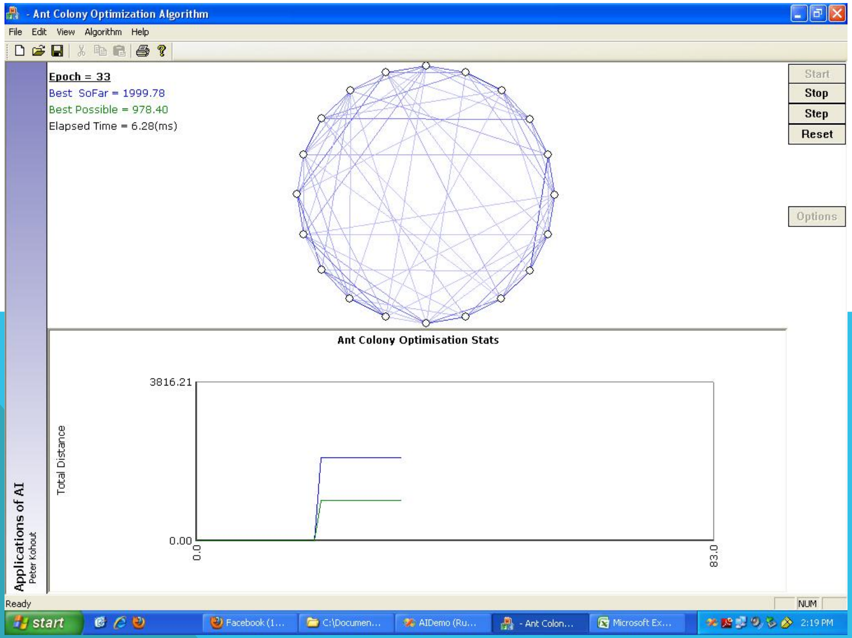
ANT COLONY OPTIMIZATION IN ACTION

- Set number of iterations the optimization will run
- Each edge gets updated with an extremely tiny, uniform level of pheromone
- Each ant is set to a random city
- Tours for each ant are built with the probability algorithm for choosing the next city
- Check to see if the best tour built is better than the current solution if one exists. If so, we make the best tour become the current solution.
- Pheromone decay algorithm is applied, keeping in mind that no ant will lay pheromone until the cycle of cities is completed.



ANALYZING THE ALGORITHM

- Using the traveling salesman problem, we can keep a constant set of cities and distances
- Analyzed the sensitivity by using source code developed by Peter Kohout called "AI Demo"
  - Allows for changing cities, ants, alpha, beta, rho values
  - Tracks cycles and computational time
- By keeping either the amount of ants or cities constant and incrementally increasing the other one, I could track changes across results



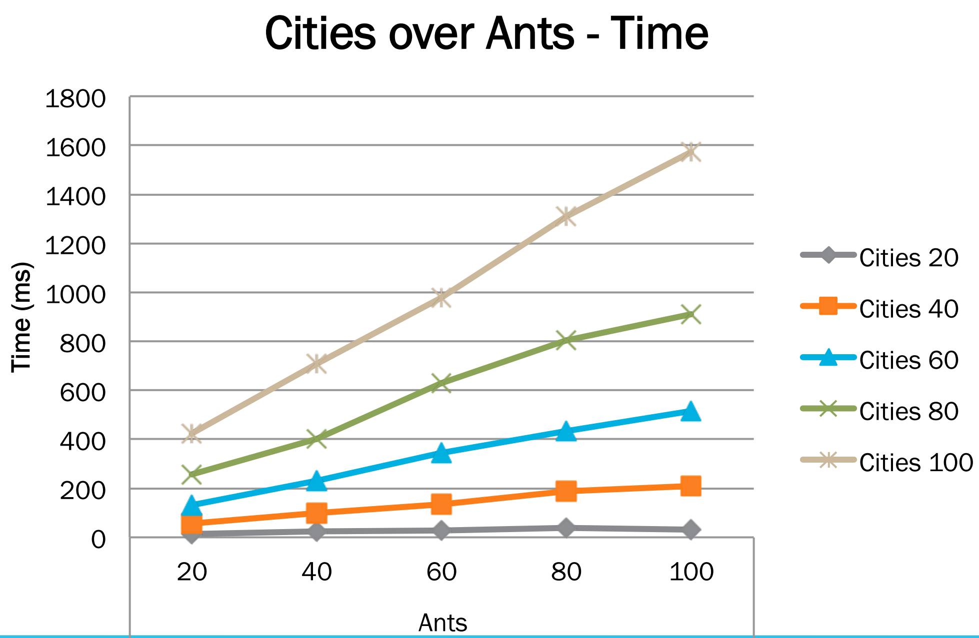
ANALYZING THE ALGORITHM

- Program Experiment Process
  - Set cities and ants to 20 each
  - Run algorithm five times, record results and determine their averages
  - Increment ants, repeat until ants reached 100
  - Reset ants to 20, increment cities to 40, restart process
  - Continued until cities reach 100
- Predictions
  - Increasing amount of ants used would require more computational time for the algorithm than increasing the amount of cities
  - Increasing the amount of ants would require less amount of cycles to come to a solution
  - Increasing the amount of cities would require more cycles to find a solution

ANALYZING THE ALGORITHM

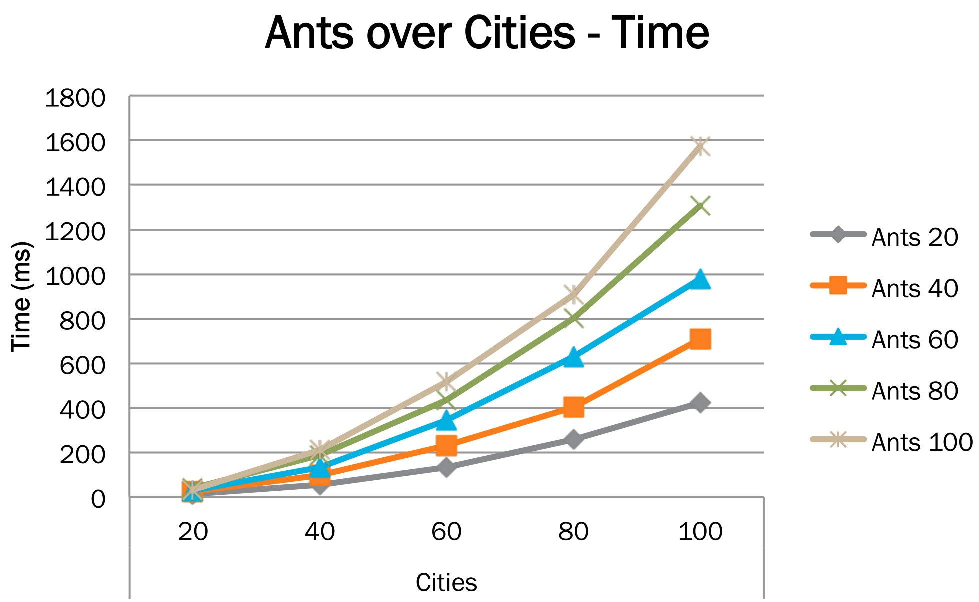
- Cities over Ants, measured by time
- Linear growth that seems to flatten near the end
- Can predict computational time based on trending growth

		Cities				
		20	40	60	80	100
Ants	20	13.39	54.78	131.58	256.976	423.86
	40	23.29	98.884	231.12	402.6	706.946
	60	28.972	133.224	344.08	628.618	976.99
	80	37.996	187.126	435.446	803.768	1309.23
	100	30.3464	209.48	515.178	910.334	1574.634



ANALYZING THE ALGORITHM

- Ants over Cities, measured by time
- Clear exponential growth
- Very low times when few ants are used, otherwise unfavorable

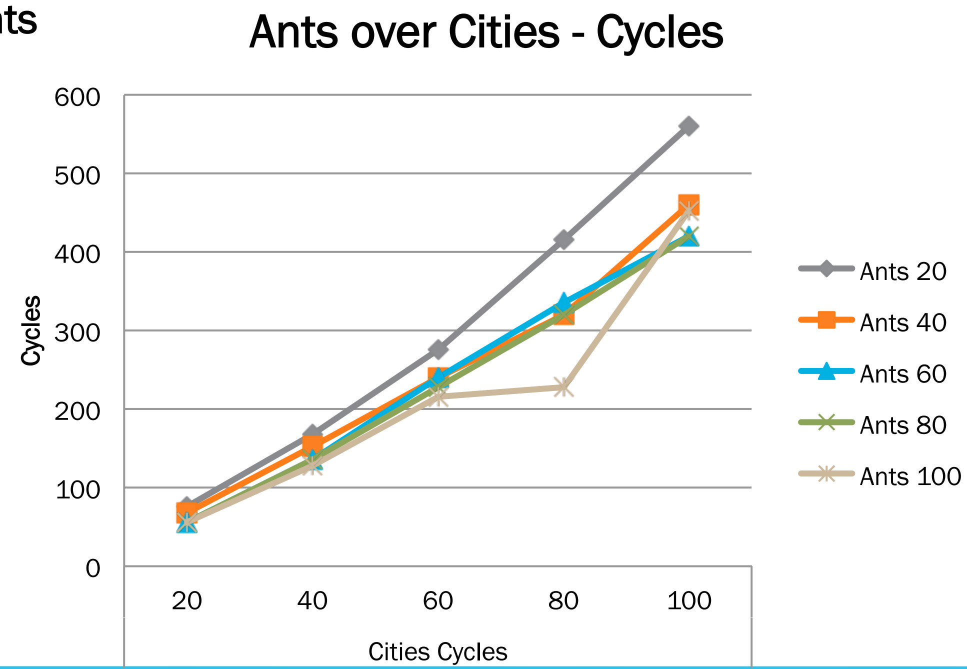


		Cities				
		20	40	60	80	100
Ants	20	13.39	54.78	131.58	256.976	423.86
	40	23.29	98.884	231.12	402.6	706.946
	60	28.972	133.224	344.08	628.618	976.99
	80	37.996	187.126	435.446	803.768	1309.23
	100	30.3464	209.48	515.178	910.334	1574.634

ANALYZING THE ALGORITHM

- Ants over cities, measured in cycles
- Very linear, although slight deviations
- Ants 20 trailing off, running out of ants

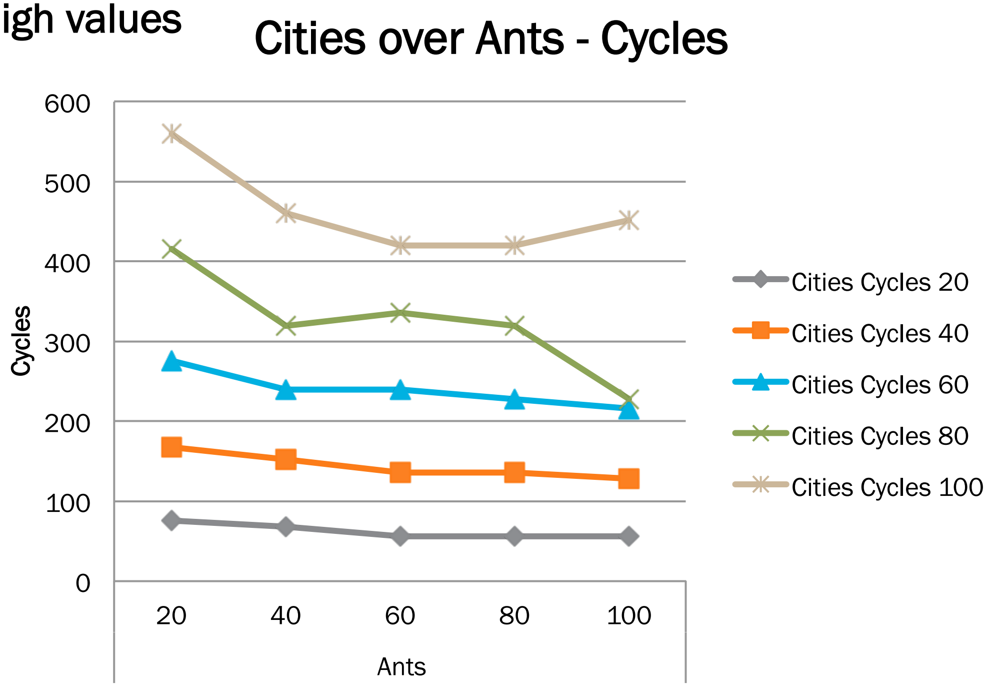
		Cities Cycles				
		20	40	60	80	100
Ants	20	76	168	276	416	560
	40	68	152	240	320	460
	60	56	136	240	336	420
	80	56	136	228	320	420
	100	56	128	216	228	452



ANALYZING THE ALGORITHM

- Cities over Ants, measured in cycles
- Consistent, no real conforming patterns
- Not enough ants at beginning causing high values

		Cities Cycles				
		20	40	60	80	100
Ants	20	76	168	276	416	560
	40	68	152	240	320	460
	60	56	136	240	336	420
	80	56	136	228	320	420
	100	56	128	216	228	452



RESULTS

- Time
  - Increasing ants causes linear growth, larger initial jumps in time
  - Increasing cities causes exponential growth, smaller initial jumps in time, yet steps get bigger with each increment
  - If computational resources and time are a problem, having more cities would not be the most efficient solution
    - Consider restructuring the problem
    - Dividing problem into smaller segments to use less cities
  - If problem is small or time and resources are no issue, use more cities
    - Represents larger solution space as opposed to sub problems
    - Small problems will still compute relatively fast

RESULTS

- Cycles
  - Increasing cities over time causes a linear growth with some deviation
    - Low ants, more cities will eventually veer off into never finding a solution
    - Possibly not enough ants to keep pheromone trail fresh
  - Increasing ants over time keeps a pretty constant growth
    - Once you hit a set amount of ants, cycles stay consistently flat – large amount of ants updating pheromone across small amounts of random paths, eventually becomes so attractive that other paths are wiped out fast
  - Increasing ants wasteful once point of consistent, flat growth is reached
  - Increasing amount of cities increases computation time AND cycles

SUMMARY

- Ant Colony Optimization is an efficient method to finding optimal solutions to a graph
- Using the traveling salesman problem and the AI demo, experiments lead to conclusions:
  - Increasing the amount of cities increases both computational time and cycles
  - Increasing the amount of ants initially uses more computational time, eventually uses less than increasing cities
  - Increase of ants also wasteful once consistent, flat measure of cycles is reached
  - More cities best used for having available computational resources and a large problem size
  - More ants best used for keeping computational resources low and efficient

WORKS CITED

Back, Thomas. "Ant Colony Optimization." *Natural Computing Group*. Web. 13 Oct. 2010. <http://natcomp.liacs.nl/NC/slides/aco.pdf>.

Dorigo, Marco, and Thomas Stützle. *Ant Colony Optimization*. Cambridge, MA: MIT, 2004. Print.

Meyer, Bernd. "Ant Colony Optimization." *Monash University*. Web. 11 Oct. 2010. <http://www.csse.monash.edu.au/~berndm/CSE460/Lectures/cse460-9.pdf>.

Algorithm images from Meyer, Bernd.